Title: LOAD-BALANCING QUEUES EMPLOYING LIFO/FIFO WORK STEALING
Inventor: Nir N. Shavit, Steven K. Heller and Christine H. Flood
Filing Date: December 6, 2002
Serial No: 10/313,878
Docket No: P8275

REPLACEMENT SHEET

Customer No. 021127

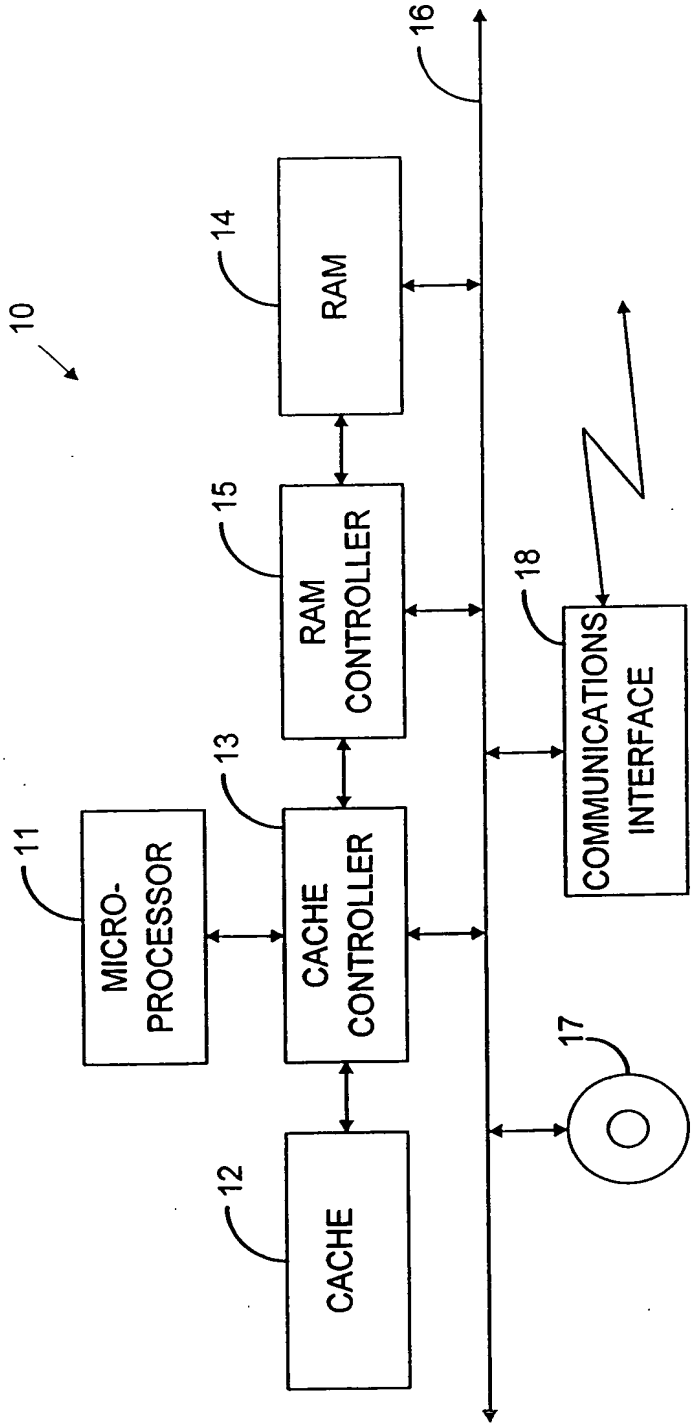


FIG. 1

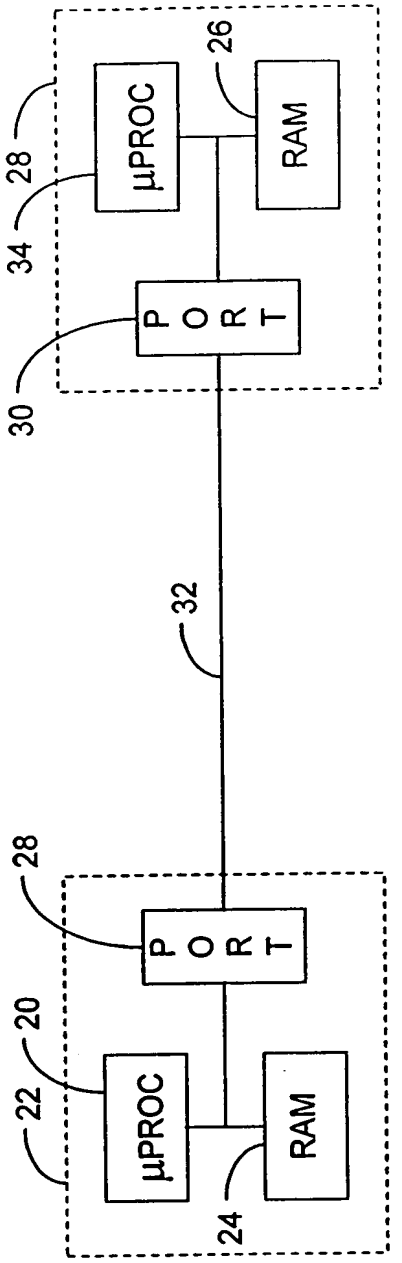

µPROC
RAM
PORT
µPROC
RAM
PORT
26
28
34
30
32
28
20
22
24

FIG. 2

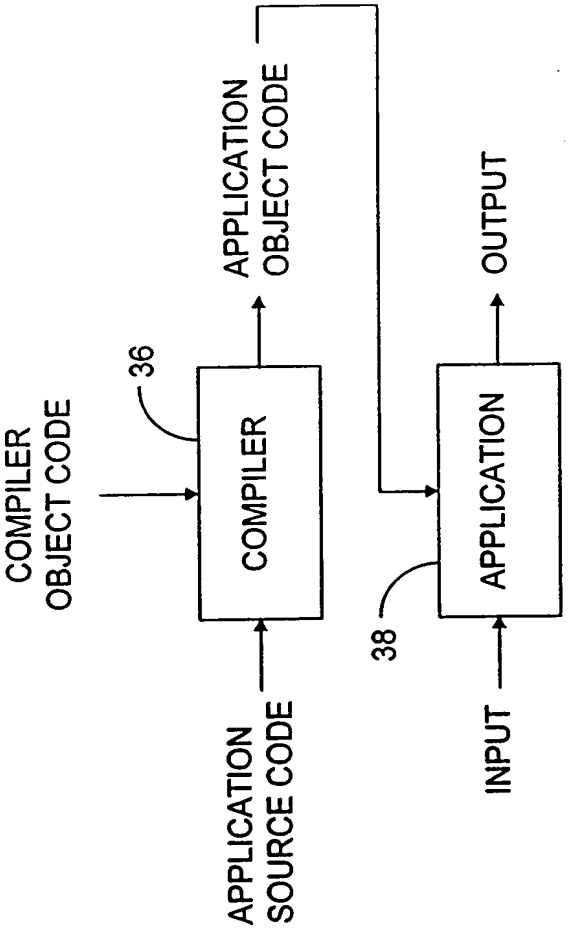

COMPILER OBJECT CODE
COMPILER
APPLICATION OBJECT CODE
APPLICATION SOURCE CODE
APPLICATION
OUTPUT
INPUT
36
38

FIG. 3

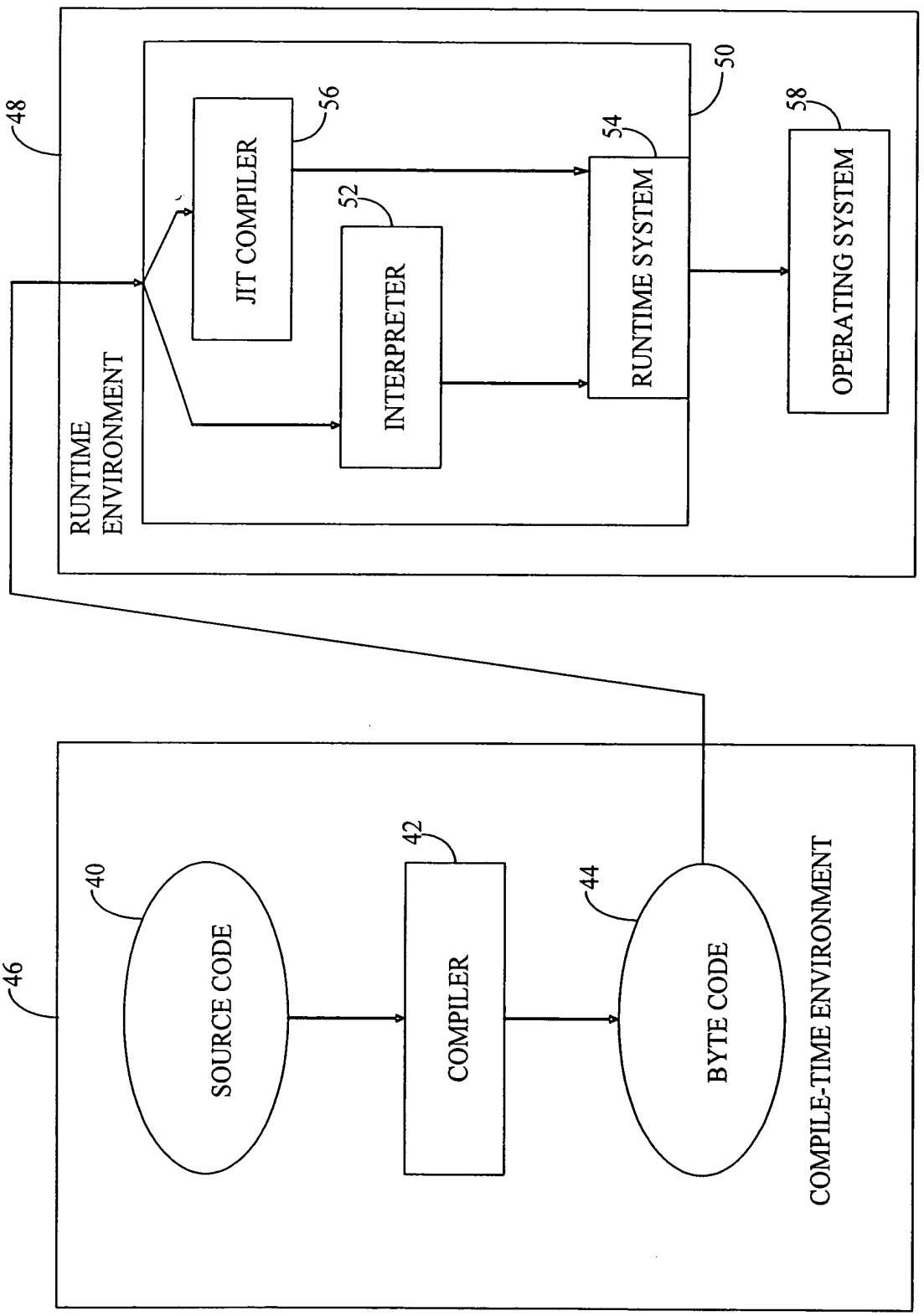

48
RUNTIME ENVIRONMENT
JIT COMPILER
56
INTERPRETER
52
RUNTIME SYSTEM
54
50
OPERATING SYSTEM
58
46
SOURCE CODE
40
COMPILER
42
BYTE CODE
44
COMPILE-TIME ENVIRONMENT

FIG. 4

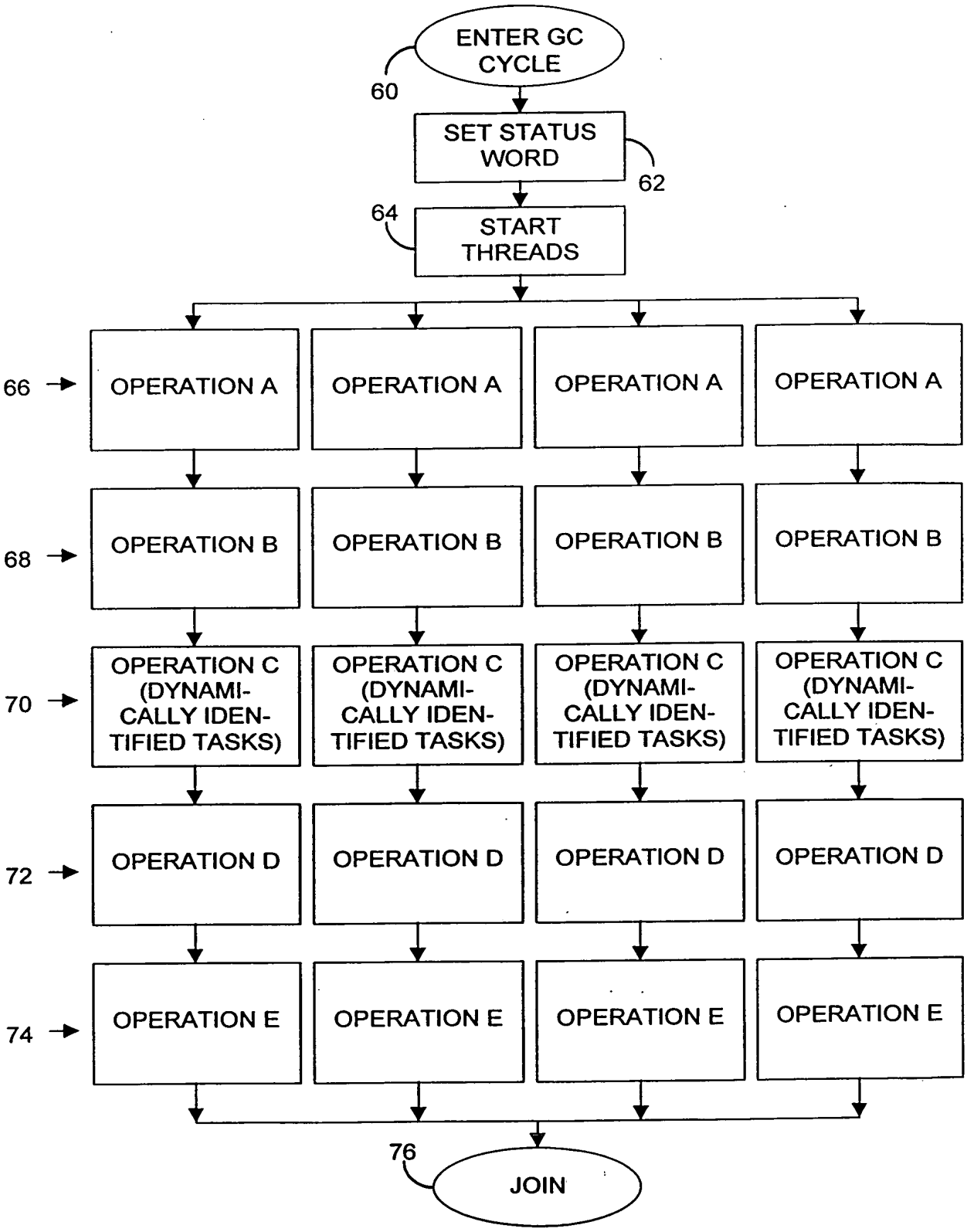

ENTER GC CYCLE
60
SET STATUS WORD
62
64
START THREADS
66
OPERATION A
OPERATION A
OPERATION A
OPERATION A
68
OPERATION B
OPERATION B
OPERATION B
OPERATION B
70
OPERATION C (DYNAMI-CALLY IDEN-TIFIED TASKS)
OPERATION C (DYNAMI-CALLY IDEN-TIFIED TASKS)
OPERATION C (DYNAMI-CALLY IDEN-TIFIED TASKS)
OPERATION C (DYNAMI-CALLY IDEN-TIFIED TASKS)
72
OPERATION D
OPERATION D
OPERATION D
OPERATION D
74
OPERATION E
OPERATION E
OPERATION E
OPERATION E
76
JOIN

FIG. 5

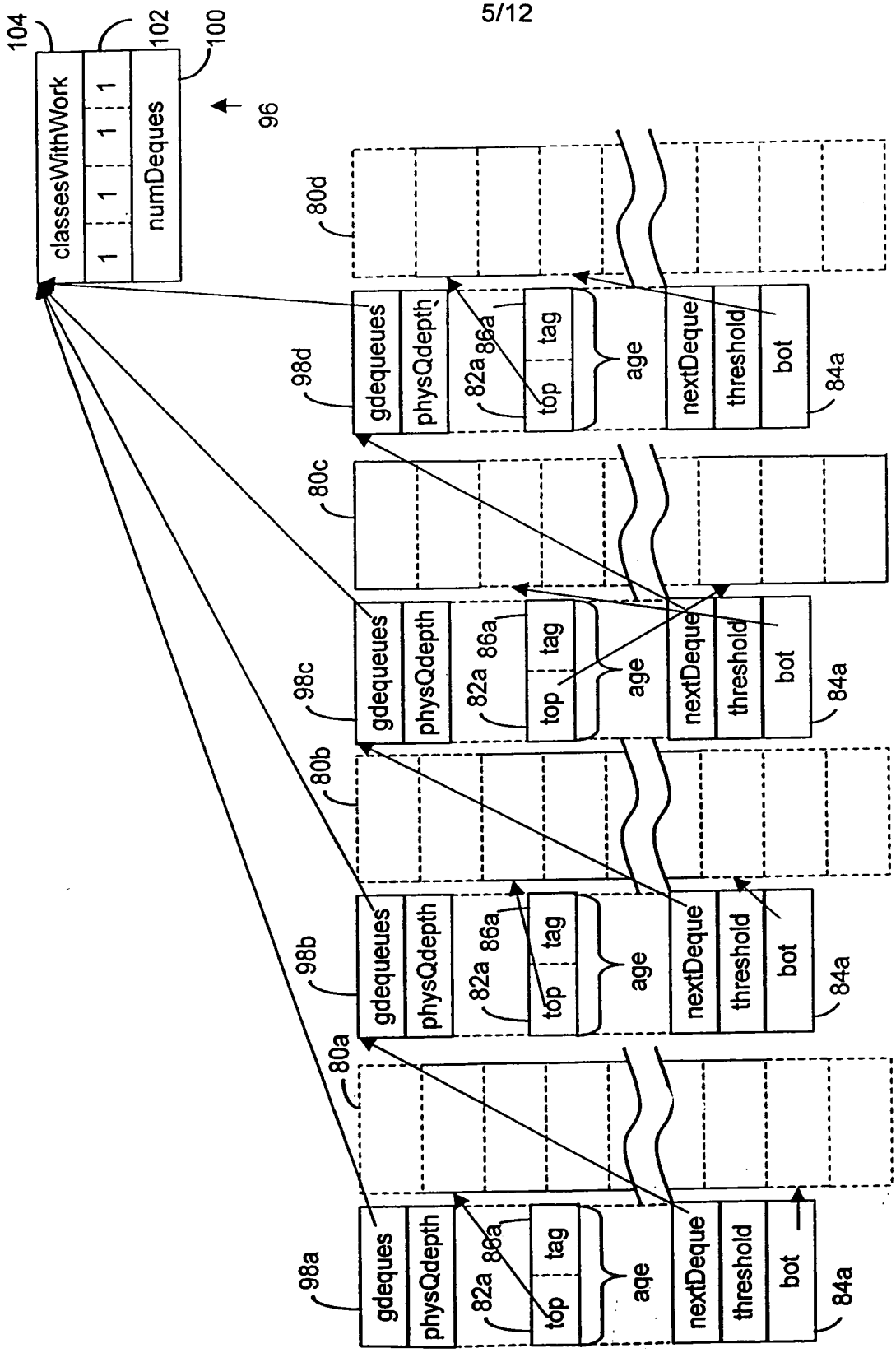

104
102
100
96
classesWithWork
numDeques
1
1
1
1
80d
98d
gdequeues
physQdepth
82a
86a
top
tag
age
nextDeque
threshold
bot
84a
80c
98c
80b
98b
80a
98a
gdeques

FIG. 6

```
1  static java_lang_Object * popTop (localDeque *dq){
2          dequeAge oldAge = dq->age;
3          unsigned int localBot = dq->bot;
4          if (localBot == oldAge.top)
5                  return NULL;
6          java_lang_Object * task = dq->objects[oldAge.top];
7          dequeAge newAge = oldAge;
8          if(++newAge.top==dq->physQdepth) newAge.top = 0;
9          dequeueAge tempAge = (dequeAge) casInt((int) newAge,
10                 (int) oldAge,
11                 (int*) &dq->age); /*atomic compare-and-swap*/
12         if(tempAge == oldAge)
13                 return task;
14         return NULL;
15 }
```

FIG. 7

```
1  static void dequePush (localDeque *dq, java_lang_Object *obj) {
2      unsigned int localBot = dq->bot;
3      dequeAge oldAge = dq->age;
4      if (dequeNumberOfElements (localBot, oldAge.top, dq) == dq->physQdepth-1) {
5          dequeOverflow(dq);
6          localBot = dq->bot;
7      }
8      setArrayElement (localBot, dq, obj);
9      if (++localBot == dq->physQdepth) localBot = 0;
10     dq->bot = localBot;
11 }

1  static int dequeNumberOfElements(unsigned int localBot, unsigned int localTop,
2              localDeque *dq) {
3      int diff = localBot – localTop;
4      if (diff < 0)
5          diff = diff + dq->physQdepth;
6          return diff;
7  }

1  static void setArrayElement(int index,
2          localDeque *dq,
3          java_lang_Object *obj) {
4      dq->objects[index] = obj;
5  }
```

FIG. 8

```
1  static java_lang_Object *dequePopWork (localDeque *dq) {
2      unsigned int localBot = dq->bot;
3      java_lang_Object *obj = NULL;
4      dequeAge oldAge, newAge;
5      int numElems = dequeNumberOfElements (localBot, dq ->age.top, dq);
6      if (numElems == 0)
7          return NULL;
8      if (numElems > dq->threshold) {
9          while ((obj == NULL) &&
10             (dequeNumberOfElements (localBot, dq->age.top, dq) > dq->threshold))
11             {
12                 obj = popTop(dq)
13             }
14         if (obj != NULL)
15             return obj;
16     }
17     if(--localBot == -1) localBot = dq->physQdepth - 1;
18     dq->bot = localBot;
19     obj = getArrayElement (localBot, dq);
20     oldAge = dq->age; /* It might have changed */
21     if (dequeNumberOfElements (localBot, oldAge.top, dq) > 0) return obj;
22     newAge.tag = oldAge.tag + 1;
23     newAge.top = oldAge.top
24     if (localBot == oldAge.top) {
25         dequeAge tempAge;
26         tempAge = (dequeAge) casInt ((int) newAge,
27              (int) oldAge,
28             (int*) &dq->age);
29     if (tempAge == oldAge) return obj;
30     }
31     dq->age = newAge;
32     return NULL;
33 }

1  static void setArrayElement(int index,
2          localDeque *dq,
3          java_lang_Object *obj) {
4      dq->objects[index] = obj;
5  }
```

FIG. 9

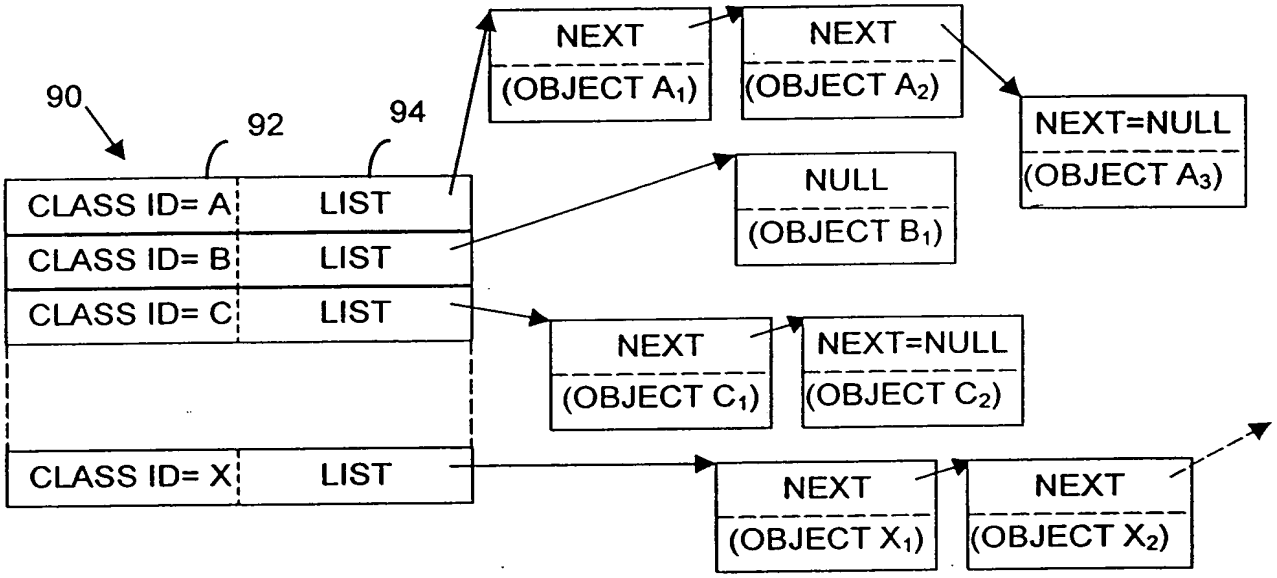
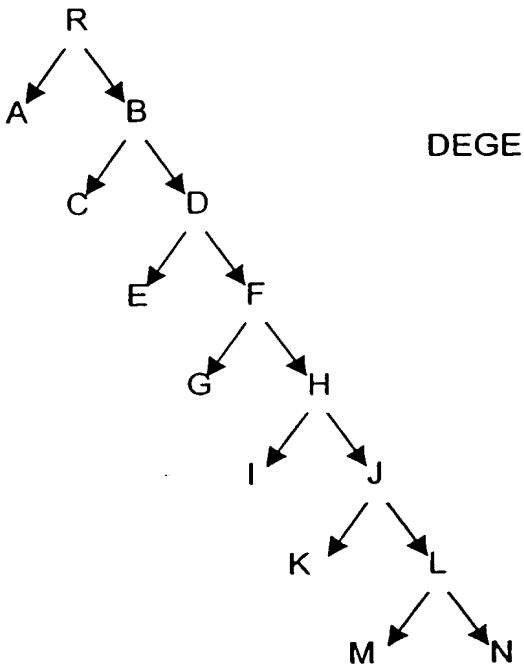

90
92
94
CLASS ID= A
LIST
CLASS ID= B
LIST
CLASS ID= C
LIST
CLASS ID= X
LIST
NEXT
(OBJECT A1)
NEXT
(OBJECT A2)
NEXT=NULL
(OBJECT A3)
NULL
(OBJECT B1)
NEXT
(OBJECT C1)
NEXT=NULL
(OBJECT C2)
NEXT
(OBJECT X1)
NEXT
(OBJECT X2)

FIG. 10



DEGENERATE TREE

FIG. 11

LIFO SEQUENCE
R
AB
ACD
ACEF
ACEGH
ACEGIJ
ACEGIKL
ACEGIKMN
ACEGIKM
ACEGIK
ACEGI
ACEG
ACE
AC
A

FIG. 12

FIFO SEQUENCE
R
AB
B
CD
D
EF
F
GH
H
IJ
J
KL
L
MN
N

FIG. 13

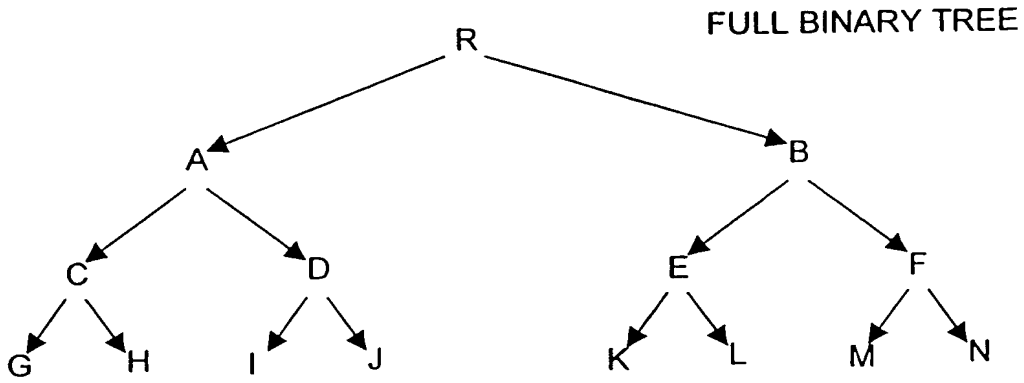


FIG.14

LIFO SEQUENCE
R
AB
AEF
AEMN
AEM
AE
AKL
AK
A
CD
CIJ
CI
C
GH
G

FIG. 15

FIFO SEQUENCE
R
AB
BCD
CDEF
DEFGH
EFGHIJ
FGHIJKL
GHIJKLMN
HIJKLMN
IJKLMN
JKLMN
KLMN
LMN
MN
N

FIG. 16

```
1  java_lang_Object *dequeFindWork(localDeque *dq) {
2      java_lang_Object *result = findWorkHelper(dq);
3      globalDeques *gdqs = dq->gdeques;
4      if (result == NULL) {
5          mark_self_inactive(dq->index, &gdqs->statusBitmap); /* You have no work */
6      }
7      while (result == NULL) {
8          if (!gdqs->statusBitmap) return NULL;  /* No one has any work. Terminate. */
9          poll(NULL, NULL, 0);
10         if (checkForWork(dq)) { /* You don't have any work, but there is some either
11                                    on the overflow queue, or in another thread's work
12                                    queue */
13             mark_self_active(dq->index, &gdqs->statusBitmap); /* Looking for work */
14             result = findWorkHelper(dq);
15             if (result == NULL) {
16                 mark_self_inactive(dq->index, &gdqs->statusBitmap);
17             }
18         }
19     }
20     return result;
21 }
```

```
1  java_lang_Object *findWorkHelper(localDeque *dq) {
2      java_lang_Object *task = findWorkInOverflowList(dq);
3      if (task == NULL) {
4          task = stealWork(dq);
5      }
6      return task;
7  }
```

```
1    static void mark_self_inactive(int self, int *pStatusBitmap) {
2            int oldValue,newValue;
3            do {
4                    oldValue = *pStatusBitmap;
5                    newValue = oldValue & (~(1<<self));
6                    newValue = casInt(newValue, oldValue, pStatusBitmap);
7            } while (newValue != oldValue) ;
8    }
```

```
1    static void mark_self_active(int self, int *pStatusBitmap) {
2            int oldValue,newValue;
3            do {
4                    oldValue = *pStatusBitmap;
5                    newValue = oldValue | (1<<self);
6                    newValue = casInt(newValue, oldValue, pStatusBitmap);
7            } while (newValue != oldValue) ;
8    }
```

FIG. 17

```
1  static java_lang_Object *stealWork(localDeque *dq) {
2      globalDeques *gdqs = dq->gdeques;
3      int degree = gdqs->numDeques;
4      int iterations = 2 * degree;
5      int i = 0;
6      while (i++ < iterations) {
7          localDeque *dqToSteal = pickQueueToStealFrom(gdqs, dq);
8          if (dqToSteal->bot > dqToSteal->age.top) {
9              java_lang_Object *task = popTop(dqToSteal);
10             if(!task) poll(NULL, NULL, 0);
11             else return task;
12         }
13     }
14     return NULL;
15 }
```

FIG. 18

```
1  static bool_t checkForWork(localDeque *dq) {
2      globalDeques *gdqs = dq->gdeques;
3      return gdqs->classesWithWork || peekDeque(dq);
4  }
```

```
1  static bool_t peekDeque(localDeque *dq) {
2      globalDeques *gdqs = dq->gdeques;
3      int degree = gdqs->numDeques;
4      int i;
5      for (i = 0; i < 2 * degree; i++) {
6          localDeque *dqToPeek = pickQueueToStealFrom(gdqs, dq);
7          if (dqToPeek->bot > dqToPeek->age.top) {
8              return TRUE;
9          }
10     }
11     return FALSE;
12 }
```

FIG. 19